Spécifiez vos éditeurs de diagrammes à l'aide de composants réutilisables

Amine El Kouhen[1,2], Cédric Dumoulin[2], Sebastien Gérard[1] and Pierre Boulet[2]

[1] Commissariat à l'Energie Atomique (CEA) - LIST,
Laboratoire d'Ingénierie dirigée par les modèles pour les Systèmes Embarqués (LISE), Gif-sur-Yvette
{amine.elkouhen, sebastien.gerard}@cea.fr
[2] Université de Lille 1, Laboratoire d'Informatique Fondamentale de Lille (LIFL) CNRS UMR 8022,
Cite scientifique - Bâtiment M3, Villeneuve d'Ascq
{amine.el-kouhen, cedric.dumoulin, pierre.boulet}@lifl.fr

Abstract

L'Ingénierie Dirigée par les Modèles (IDM) favorise l'utilisation des outils de modélisation graphique dans toutes les étapes du processus de développement logiciel. Ces outils peuvent eux-mêmes, être conçus en utilisant une approche basée sur les modèles dans des environnements appelés *outils MetaCase* (Meta-Computer-Aided Software Engineering) qui nécessitent, généralement, des efforts supplémentaires de programmation pour leur adaptation, afin d'obtenir des outils de modélisation complets pour les langages spécifiques (DSML). De plus, ces outils ont souvent des lacunes en termes de réutilisabilité. Dans ce contexte, nous proposons *MID*, un ensemble de méta-modèles qui propose la spécification rapide des éditeurs de modélisation à partir de composants graphiques réutilisables. Nous examinons par la suite, les environnements de modélisation générés à partir de ces méta-modèles.

1 Introduction

L'Ingénierie Dirigée par les Modèles (IDM) fournit un cadre méthodologique et technologique afin de promouvoir l'étude des différents aspects du système. Il n'existe actuellement aucune définition universelle d'un modèle, qui est le concept pivot de l'IDM [1]. Selon [2], les modèles sont des outils puissants pour exprimer la structure, le comportement et d'autres propriétés dans tous les domaines de l'ingénierie et de chacune des sciences dures. Dans [1], un modèle est un ensemble de faits qui caractérisent un aspect ou un point de vue sur un système pour un objectif donné.

L'IDM encourage l'utilisation d'environnements de modélisation graphique pour la manipulation des modèles. Ces outils sont très populaires dans une grande variété de domaines allant des outils de conception logiciels aux concepteurs de circuits. Plusieurs outils ont été créés pour spécifier de tels environnements pour les langages visuels. Ceux-ci comprennent MetaEdit+ [3], Meta-MOOSE [4], GME [5], AToM^3 [6] et Diagen [7].

Ces outils et technologies connaissent un grand succès dans le domaine de l'ingénierie logicielle. Toutefois, et en plus de l'intervention programmatique pour leur adaptation, plusieurs problèmes peuvent être relevés, principalement en termes de faiblesse de réutilisation et de rigidité de ces outils [8].

À un niveau d'abstraction élevé, l'étude de ces outils et plus particulièrement l'outil open source Papyrus, nous permet d'identifier des besoins en termes de critères de réutilisabilité, cohérence des données et de maintenabilité des spécifications. Le respect de ces critères nous a conduis finalement à produire un nouveau méta-outil alternatif basé sur une approche *composants*, décrite par un ensemble de méta-modèles appelés MID (Méta-modèles pour les Interfaces utilisateur et les Diagrammes), pour rapidement concevoir, prototyper et développer

des éditeurs graphiques pour un large éventail de notations visuelles. Nous nous sommes basés pour la conception de MID sur trois exigences : l'exhaustivité graphique, la facilité d'utilisation et la simplicité de la (dé)composition des éditeurs pour une meilleure réutilisation. Pour cela, nous profitons des avantages de l'IDM, de la modélisation des composants et d'un mécanisme d'héritage pour augmenter la réutilisation des composants des éditeurs. L'objectif de ce travail est la spécification et la génération d'éditeurs de diagrammes UML de Papyrus [9], à partir de composants préconfigurés réutilisables.

Dans ce but, nous considérons utile de présenter dans la deuxième section, les concepts de base des langages visuels, suivi de l'étude que nous avons faite sur des outils et des méthodes de modélisation dans la section 3. Cette dernière se termine par un résumé des questions soulevées par cette étude et une proposition de critères à respecter par notre solution. La section 4 décrit notre approche MID pour concevoir des éditeurs de diagrammes. Avant de conclure, nous validons dans la section 5, notre approche à travers deux exemples, le premier utilise le mécanisme d'héritage pour réutiliser les composants graphiques d'un diagramme et le deuxième montre la possibilité de composer quelques éléments d'un diagramme par assemblage.

2 Concepts de base des représentations visuelles

Les représentations visuelles sont parmi les plus anciennes formes de représentations de connaissance et précèdent l'apparition de l'écriture d'environ 25000 ans [10]. Les Philosophes de la science avaient étudié la linguistique et les représentations mathématiques dans l'ordre de comprendre la science, mais ce n'est que très récemment que les investigations ont commencé sur la manière avec laquelle les représentations visuelles contribuent à la compréhension. Les représentations visuelles jouent un rôle significatif dans le raisonnement scientifique [11].

Nous présentons dans cette section, les bases de cette forme de représentation dans l'optique d'extraire les concepts qui décrivent un diagramme et son éditeur. Pour cela, il sera nécessaire de définir la notion de diagramme, sa nature, ses différents types et les points essentiels pour atteindre une efficacité cognitive d'un diagramme et éventuellement de ses éditeurs sous-jacents.

2.1 Qu'est ce qu'un diagramme ?

Dans la littérature, plusieurs définitions existent pour le concept de diagramme, la définition la plus connue est celle de Kosslyn [12], Larkin [13] et Tversky [14] : *Les diagrammes sont des moyens efficaces pour présenter la pensée humaine et la résolution des problèmes. Les diagrammes sont des représentations d'information géométrique, symbolique et orienté-humain ; ils sont créés par des humains pour des humains* [15]. ils ont une petite (voire nulle) valeur dans la communication avec les machines dont la capacité de traitement visuelle est au mieux primitive [15].

Les composants élémentaires d'une représentation visuelle sont appelées les notations visuelles (ou aussi langage visuel, notation diagrammatique, notation graphique). Ces notations visuelles sont constituées d'un ensemble de symboles graphiques (**le vocabulaire visuel**), d'un ensemble de règles de composition structurant la notation visuelle (**la grammaire visuelle**) et d'une définition du sens de chaque symbole (**la sémantique**). Le vocabulaire et la grammaire visuelle forment tous les deux ce que l'on appelle la **syntaxe concrète** ou visuelle.

Les symboles graphiques sont utilisés pour représenter visuellement des constructions sémantiques, définies typiquement par un méta-modèle [16]. Les sens des symboles graphiques sont définis par leurs associations avec les concepts sémantiques qu'ils représentent. Dans les notations visuelles, une expression valide (Celle qui respecte un vocabulaire et une grammaire)

s'appelle **phrase visuelle** ou **diagramme**. Les diagrammes sont composés d'instances de symboles (tokens), arrangés et structurés selon les règles de la grammaire visuelle [17]. Une telle distinction entre le contenu (sémantique) et la forme (syntaxe : vocabulaire et grammaire) nous a permis de séparer les différentes préoccupations de notre proposition. Ces définitions sont illustrées dans la figure 1.

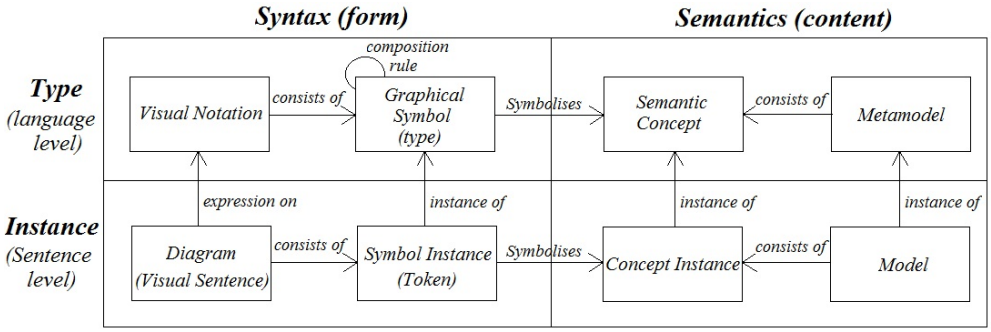

Figure 1: La nature des notations visuelles [17]

Dans ce contexte, un diagramme est défini comme un graphe planaire. Dans la théorie des graphes, la paire G = (S, A) est appelée graphe. L'élément S s'appelle Sommet (Vertex) de G et A s'appelle Arc (Edge) de G. Dans la littérature, les sommets sont aussi appelés, noeuds ou points ; les arcs sont appelés lignes ou liens [18]. Ces deux concepts et d'autres (voir section 4.2) forment les éléments de base de la grammaire visuelle d'un diagramme.

Notez qu'il est important de distinguer un modèle d'un diagramme. Un diagramme est une vue graphique sur un modèle [1]. Pour cela, il peut ne pas présenter la totalité du modèle, mais présente une partie des éléments du modèle d'une manière lisible. Plusieurs diagrammes peuvent donc être utilisées pour présenter le même modèle.

2.2 Efficacité Visuelle

Les notations visuelles sont uniquement des représentations orientées-humain : leur seul objectif est de faciliter la communication humaine et la résolution de problèmes [19]. Pour être efficaces dans ces tâches, elles ont besoin d'être optimisées pour un traitement efficace par le cerveau humain. Un diagramme est une phrase dans un langage graphique [20]. Sachant que l'objectif premier de n'importe quel langage est de communiquer, un "*bon*" diagramme est celui qui communique efficacement. L'efficacité de communication (cognitive) est mesurée par la rapidité, la facilité et la précision avec lesquelles le contenu de l'information peut être compris.

Le travail fondamental dans le domaine de la communication graphique est celui de Jacques Bertin dans la *Sémiologie des graphes* [21]. Bertin identifie huit variables visuelles élémentaires qui sont utilisées pour l'encodage de l'information (figure 2). Celles-ci sont classées en variables planaires (les deux dimensions spatiales) et les variables rétiniennes (les propriétés de l'image).

L'ensemble des variables visuelles définissent un *vocabulaire* pour la communication graphique : un ensemble de bloc de constructions atomiques qui peuvent être utilisées pour construire n'importe quelle représentation graphique. Les différentes variables visuelles sont appropriées pour encoder les différents types d'informations. Le choix des variables visuelles a un impact majeur sur l'efficacité cognitive, car il affecte à la fois la vitesse et la précision de l'interprétation [22, 23, 24].

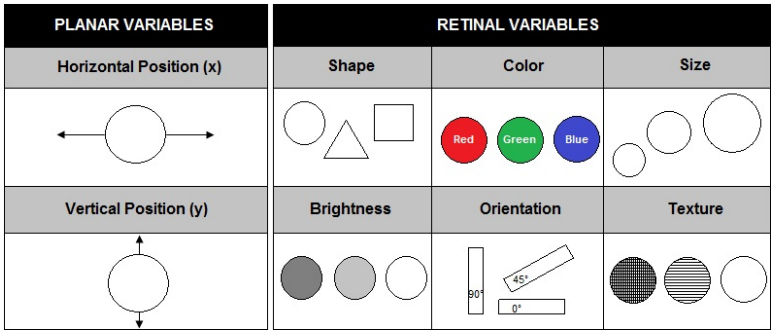

Figure 2: Variables Visuelles [21]

3 Problématique soulevée avec Papyrus

Dans le cadre de son initiative MDA (Model-Driven Architecture), l'Object Management Group (OMG)[1] - un consortium international qui représente de nombreuses institutions académiques et industrielles - a fourni une série complète de recommandations techniques normalisées à l'appui du développement basé sur les modèles. Parmi ces recommandations nous trouvons la propagation de la méta-modélisation, les transformations de modèles, les langages spécifiques (DSML) ou même les langages d'usage général (General-purpose modeling language). Un élément clé de cette dernière catégorie est UML (Unified Modeling Language), qui s'est imposé comme le langage de modélisation largement utilisé dans l'industrie et le milieu universitaire. Un certain nombre d'outils de soutien UML sont disponibles à partir de diverses sources. Toutefois, ceux-ci sont généralement des solutions propriétaires dont les capacités et disponibilités sur le marché sont contrôlées par leurs éditeurs / propriétaires. Cela peut être problématique pour les utilisateurs industriels, qui exigent des outils très spécifiques ainsi qu'un support à long terme qui, du point de vue du fournisseur, se prolonge souvent au-delà du point de viabilité commerciale. En conséquence, certains industriels sont à la recherche de solutions open source pour leurs outils UML. Pour répondre à ce besoin, un nouvel éditeur graphique, nommé Papyrus, a été acceptée par le projet Eclipse MDT en août 2008. Cet outil d'édition graphique pour UML2 est basé sur Eclipse et utilise le Framework de modélisation graphique de Eclipse (GMF)[2].

Papyrus est composé de plusieurs éditeurs (actuellement une quinzaine), principalement graphiques, complétés par d'autres types éditeurs tels que les éditeurs textuels ou arborés. L'ensemble de ces éditeurs permettent la visualisation simultanée de plusieurs diagrammes d'un modèle UML donné. La modification d'un élément dans l'un des diagrammes se reflète immédiatement dans les autres diagrammes visualisant celui-ci. Papyrus est intégré dans Eclipse comme un éditeur unique lié à un modèle UML 2. Il offre une vue principale, montrant des diagrammes de modèle et des vues supplémentaires, y compris un mode plan, une vue de propriétés et un explorateur de modèle. Les diagrammes multiples sont gérés par Papyrus plutôt que par Eclipse. Il est hautement personnalisable et permet l'ajout de nouveaux types de diagrammes développés par des technologies compatibles avec Eclipse (GEF, GMF, EMF trees...). La figure suivante illustre l'architecture générale de Papyrus.

Cependant, en spécifiant de tels diagrammes, nous avons constaté des points communs à différents niveaux. Le premier point commun constaté est celui des éléments redondants dans

[1] http://www.omg.org
[2] http://www.eclipse.org/modeling/gmf/

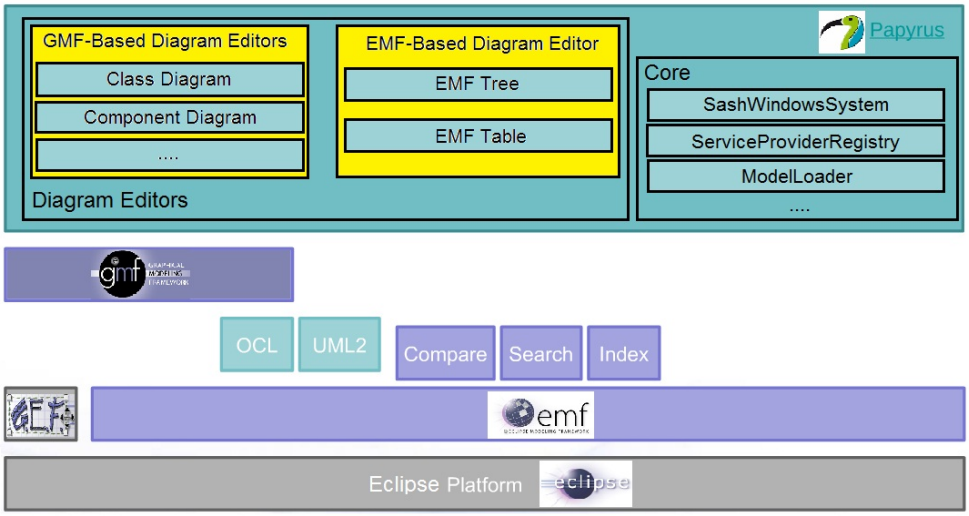

Figure 3: Architecture de Papyrus

tous les diagrammes UML, comme les éléments de commentaires (comment) ou les contraintes (constraints) qui sont présents dans tous les diagrammes Papyrus. Le deuxième point commun concerne quelques diagrammes spécifiques comme le diagramme de package qui est composé d'un sous ensemble d'éléments du diagramme de classes (package, import, merge...). L'autre point commun concerne la variation de la représentation graphique de quelques éléments. Par exemple une classe dans le diagramme de classe n'a pas la même représentation graphique que celle dans le diagramme de structure composite. Idem pour l'interface dans le diagramme de composant et dans les autres diagrammes, ou bien pour l'élément Acteur dans le diagramme de cas d'utilisation et les autres diagrammes. Ces éléments ont la même sémantique dans le modèle UML, parfois ils ont la même structure, mais ils sont représentés avec des formes différentes. Cette constatation pose une véritable problématique de réutilisation lors de la spécification des diagrammes.

Pour expliquer ces problèmes nous avons effectué une évaluation des technologies utilisées actuellement pour spécifier les diagrammes dans Papyrus et d'autres techniques et outils qui permettent de le faire dans l'état de l'art. Il s'avère que la cause principale de telles faiblesses est l'absence de mécanisme permettant la réutilisabilité dans ce genre de technologie. Ceci se traduit par des copies manuelles dans tous les diagrammes, augmentant ainsi le risque d'erreur, les problèmes de cohérence, la redondance au niveau de la spécification et la difficulté de la maintenance.

Pour appréhender ces problématiques, nous vous proposons un aperçu de notre évaluation des outils et technologies de spécification des diagrammes. Les méthodes de spécification de diagrammes ont été largement discutée dans [8, 16, 25]. Nous pouvons les résumer dans les catégories suivantes :

1. Spécification par le code. Comme c'est le cas pour GEF, UMLet [26] ou Graphiti. Cette technologie est la plus utilisée, elle permet de construire des représentations graphiques à partir d'une description programmatique (code).

2. Spécification basée sur les langages propriétaires. Cette catégorie d'outils utilise des méta-

langages de description pour spécifier les diagrammes, c'est le cas de MetaEdit+, qui utilise le langage GOPPRR méta-modélisation [3] et GME [5], basé sur une méta-description pour définir les syntaxes abstraites et concrètes d'un domaine spécifique.

3. Spécification basée sur la grammaire graphique. Ce genre d'outil est basé sur la définition de la grammaire graphique et permet de spécifier des diagrammes à partir de cette grammaire visuelle (par exemple Diagen [27], AToM^3 [28]).

4. D'autres outils permettent la spécification de diagrammes ; ils sont exclusivement des dessinateurs graphiques c'est-à-dire qu'ils ne se préoccupent que de l'aspect graphique sans associer les éléments du diagramme à une sémantique (Microsoft Visio et Dia).

5. Il existe d'autres méthodes et outils, qui ont récemment adopté des approches dirigées par les modèles ; on distingue deux grandes catégories dans cette méthode de spécification :

   - Des outils basés sur les profils UML comme Papyrus [9], IBM RSA et MagicDraw, qui proposent de créer des représentations visuelles pour les méta-classes des profiles (mécanisme des stéréotypes UML).
   - Outils basés sur un langage spécifique. Les langages spécifiques de modélisation (DSML) sont plus spécifiques aux exigences du domaine, ils offrent aux utilisateurs des concepts propres à leur métier et leur savoir-faire. GMF, TopCased et Obeo Designer sont des exemples de ce type de spécification. Ils permettent de décrire (via des modèles) la syntaxe concrète et de l'associer à un méta-modèle spécifique à un domaine. Ce genre de technologie, notamment GMF, est celle utilisée dans papyrus pour plusieurs raisons (facilité de gérer un modèle par rapport au code, technologie open-source, facilité d'intégration avec l'écosystème Eclipse ...). Cependant, ces outils nécessitent encore des interventions programmatiques pour l'adaptation. De nombreuses autres lacunes sont détectées, principalement en termes de faiblesses de réutilisation et de rigidité des outils [8, 16].

Cet article se focalise sur le critère de la réutilisabilité que nous jugeons très utile dans le cadre de la spécification des diagrammes. Parmi les formes de réutilisation qui existent, nous pouvons citer la séparation des préoccupations, l'héritage et la surcharge.

La plupart des méthodes de spécification mentionnées ci-dessus mélangent les préoccupations. La forme la plus commune de ce mélange est celle de la forme et du contenu (représentations visuelles et sémantiques). Par exemple, dans le cas de spécification d'un diagramme à l'aide de GME ou MetaEdit +, ces outils permettent de créer des concepts spécifiques et leurs représentations associées dans le même référentiel. Ceci introduit un couplage plus fort entre les aspects sémantiques et graphiques. Le même problème est observé avec Obeo Designer, qui permet d'associer des concepts graphiques aux éléments du méta-modèle du domaine, dans le même modèle. Une autre forme de mélange de préoccupations se situe entre le vocabulaire visuel et la définition de la grammaire. En effet, la plupart des outils qui offrent la séparation de la partie graphique et de la sémantique, comme GMF, Topcased ou même des standards comme Diagram Definition (DD) [29], ne proposent pas la séparation des préoccupations au niveau de la syntaxe graphique : entre le vocabulaire (formes, couleurs, styles .. .) et la grammaire (structure et composition des représentations).

Nous avons relevé un autre problème dans GMF ou Obeo Designer : le peu de possibilités (voire l'absence) de réutilisation des éléments de spécification, ce qui se traduit généralement par une redondance des spécifications, au niveau modèle et des classes au niveau du code généré.

La conception de Papyrus a fait apparaitre un besoin important en termes de réutilisabilité dans la définition des diagrammes. Il serait bien de pouvoir décrire une bibliothèque de syntaxe concrète indépendante de la sémantique, de réutiliser si nécessaire cette description dans différents diagrammes et de pouvoir factoriser les points communs de la syntaxe concrète dans des définitions communes et les variantes dans des définitions spécifiques basées elles aussi sur la définition commune. Ceci nous permettra de définir chaque diagramme indépendamment de l'autre avec la possibilité de réutiliser et redéfinir les éléments en commun.

4 Solution : Séparer Pour Mieux Réutiliser

Le but de notre travail consiste à concevoir des diagrammes et de permettre de réutiliser quelques parties de cette conception. Pour cela, nous proposons d'utiliser une approche dirigée par les modèles, pour assurer l'indépendance de la technologie, faciliter la maintenance et permettre une bonne pérennité. Pour améliorer la réutilisabilité, nous proposons une approche basée sur les composants. Cette approche a pour but de bénéficier des avantages de l'encapsulation (facilité de maintenance et de composition) et les avantages de l'interfaçage (mécanisme de nommage des interfaces). De plus, notre approche permet de réutiliser par héritage : un composant peut hériter d'un autre composant et il peut aussi redéfinir certaines de ses caractéristiques (style, structure, assemblage ...), l'héritage d'un composant se fait par la réutilisation de son interface en respectant le nommage de celle-ci. Le premier avantage de cette proposition est l'indépendance de la sémantique qui permet d'assembler les éléments graphiques différemment d'un diagramme à un autre. Un autre avantage qui se montre pertinent est celui de la facilité de maintenance de cette spécification. En évitant la redondance et les doublons au niveau du code généré, nous pouvons améliorer la maintenabilité de la spécification du diagramme, mais aussi en proposant un mécanisme qui nous permet de répercuter toute modification d'une définition sur tous les endroits utilisant cette définition. L'approche composant se montre prometteuse dans ce cadre. En définissant la composition de la structure d'un élément graphique et l'encapsulation de ses interfaces, on peut arriver à faciliter la maintenance et la modification des spécifications.

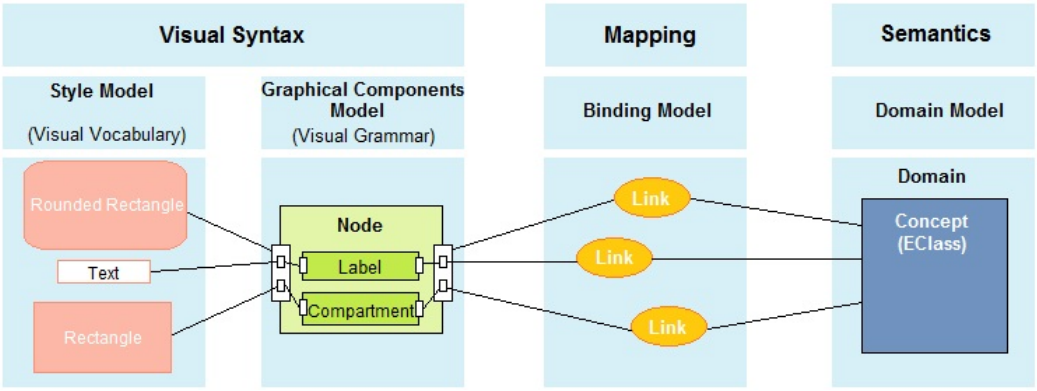

Figure 4: MID: les différents artéfacts concernés

Pour une meilleure réutilisabilité, nous proposons aussi une séparation claire des préoccupations. La figure 4 montre le lien entre les différents artefacts concernés. Tout d'abord, nous séparons le contenu du domaine (**la sémantique**) de la forme (**la syntaxe visuelle**) d'un diagramme à un niveau d'abstraction élevé (au niveau langage ou méta-modèle). Le contenu (la sémantique)

d'un diagramme est hors portée de notre travail, il est largement traitée dans les outils et les technologies comme EMF / Ecore [30]. La forme est séparée en deux parties: le vocabulaire visuel (différentes variables de forme, couleur, taille ...) et la grammaire graphique qui décrit les règles de composition des représentations visuelles. Le lien entre la syntaxe et la sémantique est également précisé dans un modèle d'association/assemblage. Ainsi, notre proposition est faite de plusieurs méta-modèles, chacun est utilisé pour décrire une préoccupation : un méta-modèle de composants, un méta-modèle de grammaire visuelle, un méta-modèle de style et un méta-modèle d'assemblage. L'ensemble de ces méta-modèles forme ***MID*** pour **M**éta-modèles pour les **I**nterfaces utilisateur et les **D**iagrammes.

4.1 Méta-modèle des Composants

Le méta-modèle des composants (figure 5) est le coeur (core) de notre ensemble de méta-modèles. Il sert principalement à définir la notion de composants qui est utilisée dans les autres méta-modèles. Un composant peut avoir des interfaces (dans notre contexte, il y'a trois types d'interfaces : interfaces de domaine, interfaces de style et les interfaces des événements) et des associations entre ces interfaces. Les interfaces sont utilisées comme points d'attache entre les (sous-)composants et d'autres préoccupations (la sémantique, les événements, les variables visuelles ...).

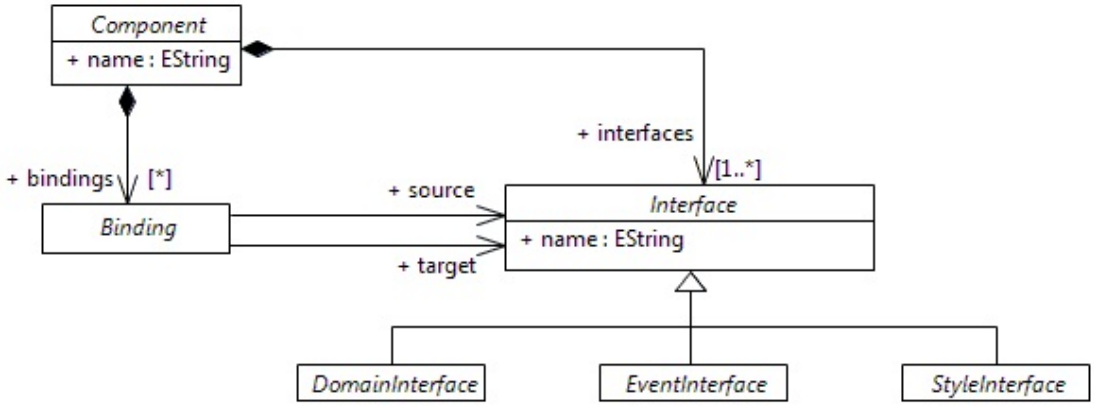

Figure 5: Méta-modèle des Composants

Notre objectif est de spécifier et de composer les éléments de l'éditeur de diagramme avec une approche basée sur les caractéristiques des composants. Nous suivons une approche à base de modèles pour la modélisation des composants du diagramme pour résoudre deux problèmes : l'hétérogénéité des composants et leurs techniques de composition.

4.2 Grammaire Visuelle

La grammaire visuelle permet de décrire la structure des éléments des diagrammes. Cette description est hiérarchique : un élément racine peut contenir d'autres éléments. Nous proposons deux types principaux d'éléments : les sommets ou noeuds (Vertex) pour représenter les éléments complexes des diagrammes et les Arcs (Edge) pour représenter les liens entre les éléments complexes.

Le concept de Vertex est l'abstraction d'un noeud et la formalisation d'une figure [31], il se compose de noeuds principaux (noeuds supérieurs), de sous-noeuds (noeuds appartenant à d'autres noeuds) et de noeuds de bordures (noeuds qui peuvent être apposées sur d'autres noeuds). Les compartiments sont des noeuds spécifiques qui servent à structurer et contenir d'autres éléments. Une étiquette (Label) est un noeud qui permet de spécifier des éléments textuels de noeuds et de leur associer des accesseurs (getters et setters). Cela permet de synchroniser le modèle de données avec la valeur du texte représenté. Un lien (Edge) est une connexion entre les éléments de diagramme. La relation entre l'élément source et celui de destination est spécifiée sémantiquement (dans le modèle de domaine). Tous ces éléments sont des composants, ils héritent directement ou indirectement du concept Component vu précédemment.

Le méta-modèle dans la figure 6 représente les concepts principaux d'un diagramme. Il permet de concevoir tous les langages hybrides visuels [25], que nous utilisons dans le domaine de l'IDM comme UML, BPMN, les graphes d'état ...

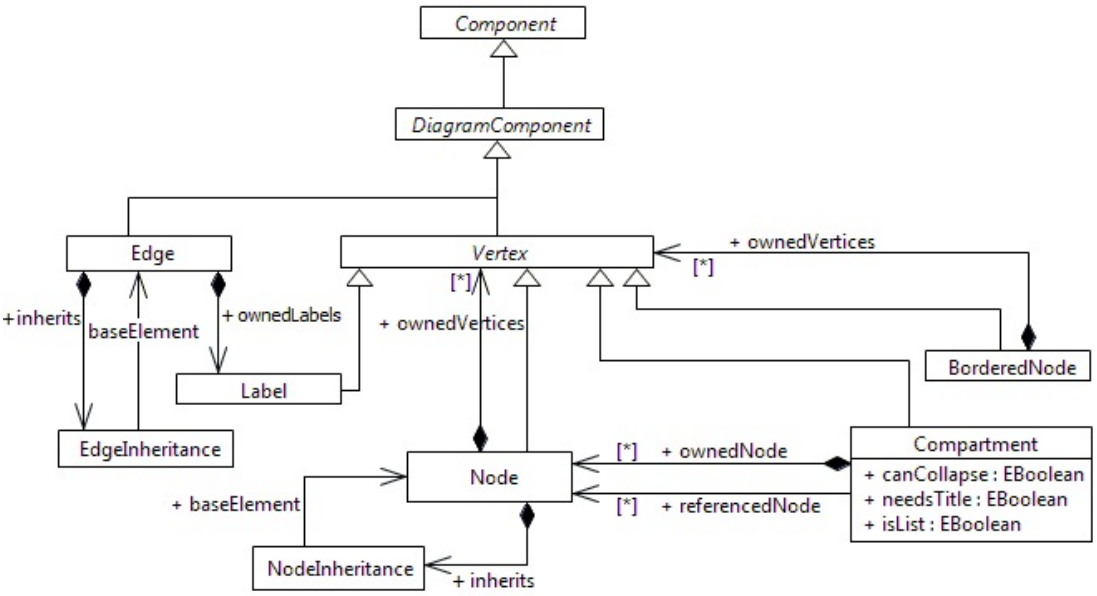

Figure 6: Eléments du diagramme

Les liens et les noeuds ont la capacité d'hériter d'autres liens et noeuds. Ils peuvent aussi redéfinir la composition, le style et le comportement des composants spécialisés. Cette fonctionnalité permet la réutilisation des composants ainsi que la création d'autres composants dérivés.

Les composants graphiques (éléments de la grammaire visuelle) sont associés à un vocabulaire visuel dont nous décrivons le méta-modèle dans la section suivante.

4.3 Vocabulaire visuel : Variables Visuelles

Le vocabulaire visuel permet de décrire les symboles graphiques des éléments de diagramme. Cette description est composée de différentes variables visuelles ; nous les regroupons toutes dans le concept de *Style* (figure 7), représentant la forme, la couleur, la taille... Tous les composants du diagramme sont associés aux vocabulaires visuels représentés via leurs interfaces

de style. Comme toutes autres caractéristiques des éléments graphiques, cette relation peut être réutilisée et surchargée par l'intermédiaire du mécanisme proposé d'héritage. Les styles de Vertex sont caractérisés par l'attribut de structure (Layout), qui représente les différentes règles d'arrangement de la figure hôte.

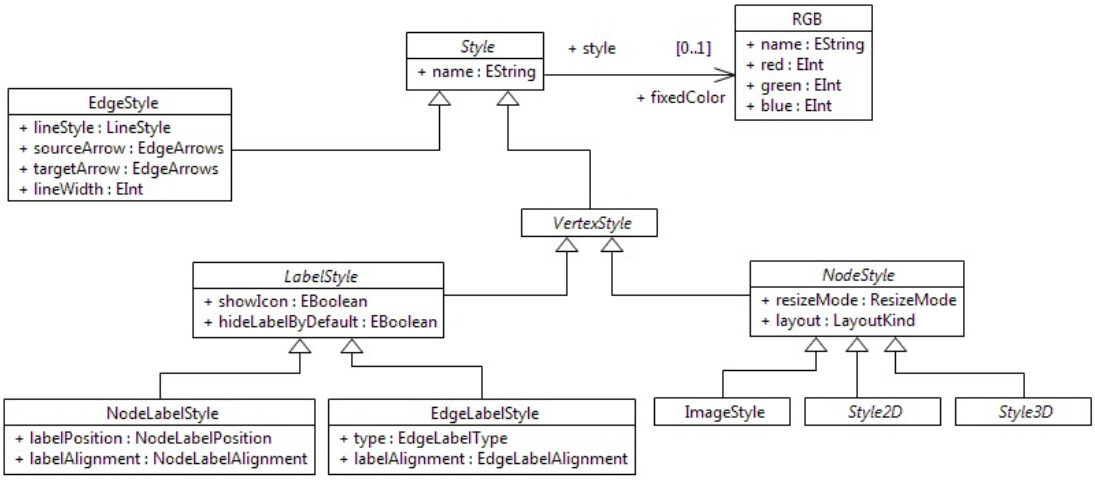

Figure 7: Description des variables visuelles (Styles)

Le style Vertex (**VertexStyle**) se décompose en deux grandes catégories : style de noeud (**NodeStyle**) et style d'étiquette (**LabelStyle**). Les styles de noeud représentent les formes (des figures 2D et 3D et des représentations en images). Notre méta-modèle propose une dizaine de formes de base. Il permet aussi à l'utilisateur de définir ses propres formes grâce aux concepts de **PolygonStyle**. Dans ce concept, l'utilisateur définit sa forme en décrivant un polygone (ensemble de points). Il est aussi possible de créer des styles personnalisés avec des images ou directement avec une implémentation dans le code. Les styles d'étiquettes précisent l'alignement des étiquettes, leurs positions et leurs types.

4.4 Association sémantique et Assemblage du Diagramme

Le méta-modèle d'association (figure 8) permet d'associer les différents éléments graphiques (noeuds et liens) aux éléments sémantiques du domaine. Un élément graphique peut être associé à un ou plusieurs éléments sémantiques. Cela se fait grâce aux concepts **NodeBindingSet** et **EdgeBindingSet**, contenant chacun un ensemble d'associations **NodeBinding** ou **EdgeBinding**.

Dans la version actuelle, la description des associations sert comme point d'entrée à la description complète du diagramme. Ceci est représenté par l'élément **Diagram** qui contient tous les ensembles d'associations. Cette approche nous permet de réutiliser des représentations graphiques dans des diagrammes différents avec des sémantiques différentes. Par exemple, dans un diagramme de classes UML, il est possible d'utiliser la même représentation pour les deux concepts *Class* et *Interface*. Nous utilisons cette fonction pour augmenter la réutilisabilité des composants graphiques.

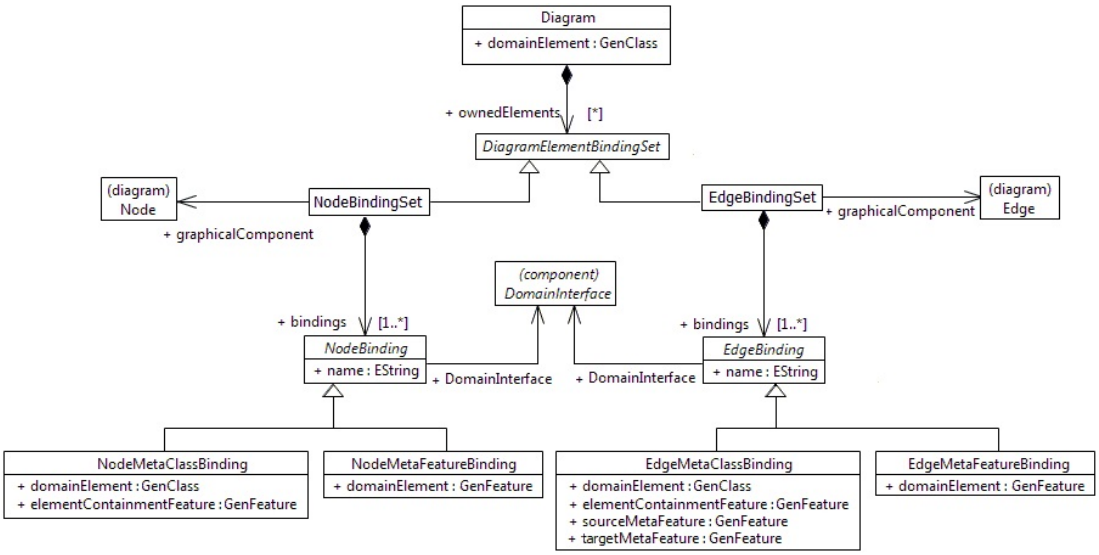

Figure 8: Assemblage de l'éditeur de Diagramme

5 Validation

Pour valider notre proposition, nous avons développé une chaîne de transformations produisant des éditeurs opérationnels. Pour l'implémentation, nous ciblons le framework GMF et son méta-modèle gmfgen. Nous allons illustrer les avantages de notre approche, sur deux exemples : le premier en spécifiant l'élément *Classifier* de UML pour montrer la réutilisabilité par héritage, le deuxième exemple pour montrer la réutilisation par assemblage. Pour ce dernier, nous avons choisi l'exemple de réutilisation de l'élément commentaire (Comment) dans tous les diagrammes conçus. Nous avons aussi validé notre approche sur d'autres diagrammes que nous ne décrivons pas dans cet article.

5.1 Réutilisation par Héritage

Nous avons choisi pour cet exemple le cas du concept *Classifier* d'UML, ce concept abstrait est l'élément de base de plusieurs concepts (*Class*, *Interface*, *Composant*...). Nous voulons spécifier l'aspect graphique de l'élément *Classifier*, et d'utiliser l'héritage et la surcharge pour spécialiser cette spécification. L'élément de base *Classifier* se compose graphiquement d'un label suivi d'un compartiment qui contient des propriétés. Pour spécifier l'élément *Composant*, il suffit d'hériter de *Classifier* et d'ajouter à sa structure, un noeud de bordure qui représentera les ports du composant. Pour Spécifier les représentations graphiques des éléments *Class* et *Interface*, il suffit d'hériter de *Classifier* en ajoutant à sa structure deux autres compartiments, un pour les opérations et l'autre pour les classes imbriquées (nested classifier). Dans cet exemple ; et par simplicité ; nous faisons hériter la définition graphique de l'interface de la classe pour montrer la similitude graphique entre les deux concepts. La figure suivante illustre cet exemple.

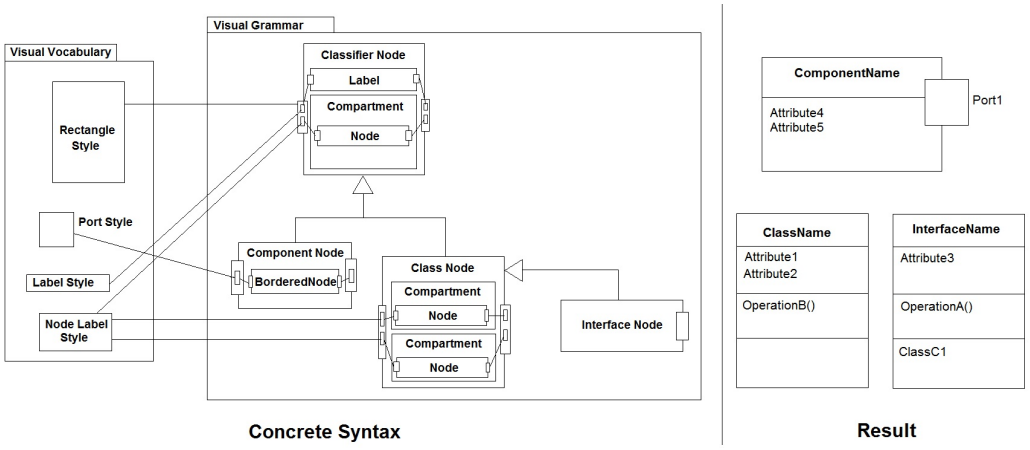

Figure 9: Réutilisation par Héritage

5.2 Réutilisation par Assemblage

Nous avons choisi l'exemple des commentaires qui sont utilisés dans tous les diagrammes. Dans ce cas, il suffit de spécifier le composant graphique d'un commentaire (Comment) et de l'assembler dans toutes les définitions des diagrammes. La figure suivante montre cet exemple.

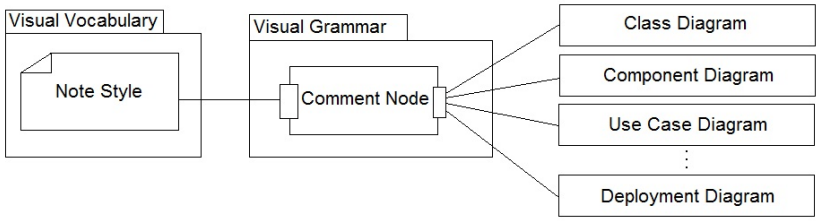

Figure 10: Réutilisation par Référence

5.3 Chaîne de transformation MID − > GMF générateur de code (gmfgen)

Pour valider rapidement notre approche, nous avons développé une chaîne de transformation pour générer le code java pour les éditeurs de diagramme, en ciblant le Framework GMF. La chaîne de transformation nous permet de passer d'un espace de travail de modélisation à l'autre. Chaque espace de travail est à un niveau d'abstraction et de détail différent de l'autre. Les Modèles intermédiaires sont décrits par des méta-modèles et servent à ajouter des détails techniques et des technologies requises pour la génération de code. Nous avons pris soin de faire apparaître le plus tard possible les détails technologiques propre à la cible visée. Ces détails n'apparaissent que dans les derniers modèles de la chaîne.

Cette chaîne de transformation nous permet de générer 100% du code opérationnel des éditeurs de diagrammes. Le développeur n'a plus qu'à exécuter l'application. L'éditeur généré permet de manipuler des concepts spécifiques au domaine avec des représentations graphiques

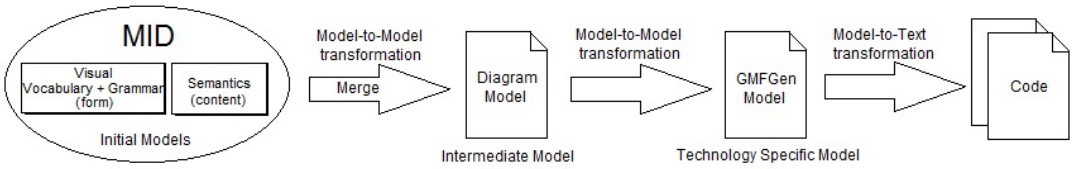

Figure 11: Chaîne de transformation MID − > GMF

spécifiées au niveau du modèle. Actuellement, notre approche permet la réutilisation au niveau de la conception des diagrammes. Cependant, la technologie cible choisie, ici GMF, ne permet pas une réutilisation du code généré. Celui-ci reste dupliqué, mais avec une définition unique au niveau de nos méta-modèles.

6 Conclusion et Perspectives

Dans cet article, nous présentons une approche basée sur les modèles et les composants pour la conception des éditeurs graphiques de diagramme. Ceci permet la spécification rapide des éditeurs graphiques de diagrammes à un niveau d'abstraction élevé, afin de modéliser, de réutiliser, de composer et de générer du code. Dans notre proposition, nous nous concentrons sur le concept de composant pour décrire et ensuite assembler les nouveaux concepts des langages visuels (langage diagrammatique). Tout d'abord, nous présentons les définitions et les fondements théoriques des représentations visuelles et l'approche basée sur la méta-modélisation des composants tout en positionnant notre travail par rapport aux différents outils et technologies disponibles dans l'industrie et dans la littérature.

Dans notre approche, nous encourageons l'utilisation de composants réutilisables (avec la composition, l'encapsulation et l'héritage ...) et la séparation forte des préoccupations (domaine, élément graphique, les variables visuelles). Cela augmente la réutilisabilité des éditeurs de diagrammes et apporte les avantages du paradigme IDM comme la vérification/validation des modèles ou la possibilité de choisir les technologies "cibles" grâce à des techniques de transformation de modèles. Dans MID, nous résolvons certains problèmes identifiés dans les outils et méthodes existants dans l'industrie comme dans la littérature. Par exemple, la spécification à un haut niveau d'abstraction, sans la nécessité d'une intervention programmatique manuelle, la séparation des préoccupations, l'efficacité graphique et enfin la réutilisation des éditeurs, qui faisait partie des problématiques majeures de notre travail de recherche. Pour valider notre approche, nous avons développé une chaîne de transformation visant la technologie GMF (GMFGen), qui permet la génération de code fonctionnel de l'éditeur de diagrammes. Cela nous permet de concevoir avec succès des diagrammes en réutilisant des composants existants et de générer leur code. Nous avons validé notre approche sur plusieurs diagrammes.

Notre approche présente plusieurs avantages : Tout d'abord, grâce à la réutilisation de composants sous forme de modèles : les modèles sont théoriquement plus faciles à comprendre et à manipuler par les utilisateurs finaux, ce qui correspond à un objectif de l'IDM. Ensuite, en gain de productivité : il est possible de constituer des librairies de composants, puis de construire son diagramme par assemblage de ces composants. Les éléments graphiques et les éléments du domaine peuvent évoluer séparément.

Nous pouvons dire que notre approche ouvre une nouvelle voie qui se montre prometteuse pour une plus large utilisation des outils de modélisation et de génération automatique d'applications. Par rapport aux technologies actuelles de développement, les promesses de cette

approche sont importantes, grâce à la possibilité de créer des applications complexes en assemblant de simples fragments de modèles/composants existants et surtout la possibilité pour les non-informaticiens, experts dans leurs domaines d'activité, de créer leurs propres applications à partir d'une description de haut niveau en utilisant un formalisme adapté, facile à comprendre et à manipuler pour eux.

Dans l'état actuel de nos recherches, de nombreuses études sont encore nécessaires pour atteindre une génération complète d'outils de modélisation. Tout d'abord, nous devons finaliser la description et la génération de tous les éditeurs graphiques de Papyrus avec notre approche. Enfin, nous devons définir d'autres méta-modèles qui permettent la description des autres parties de ces outils (styles dynamiques, interactions et comportements des éléments graphiques...) en suivant la même approche de la réutilisation de composants et d'héritage.

References

[1] Jean-Marc Jezequel, Benoit Combemale, and Didier Vojtisek. *Ingénierie Dirigée par les Modèles : des concepts à la pratique*. Editions ellipses, Paris, 2012.

[2] Jonathan Sprinkle, Bernhard Rumpe, Hans Vangheluwe, and Gabor Karsai. Metamodelling: state of the art and research challenges. In *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems*, MBEERTS'07, pages 57–76, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Steven Kelly, Kalle Lyytinen, and Matti Rossi. Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In Panos Constantopoulos, John Mylopoulos, and Yannis Vassiliou, editors, *Advanced Information Systems Engineering*, volume 1080 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin Heidelberg, 1996.

[4] R.I Ferguson, N.F Parrington, P Dunne, C Hardy, J.M Archibald, and J.B Thompson. Metamoose: an object-oriented framework for the construction of case tools. *Information and Software Technology*, 42(2):115 – 128, 2000.

[5] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing domain-specific design environments. *Computer*, 34(11):44 –51, nov 2001.

[6] Juan de Lara and Hans Vangheluwe. Atom^3: A tool for multi-formalism and meta-modelling. In Ralf-Detlef Kutsche and Herbert Weber, editors, *Fundamental Approaches to Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2002.

[7] M. Minas and G. Viehstaedt. Diagen: a generator for diagram editors providing direct manipulation and execution of diagrams. In *Visual Languages, Proceedings., 11th IEEE International Symposium on*, pages 203 –210, sep 1995.

[8] Amine El kouhen, Cédric Dumoulin, Sebastien Gérard, and Pierre Boulet. Evaluation of modeling tools adaptation. Technical report, CNRS, 2011. `http://hal.archives-ouvertes.fr/hal-00706701`.

[9] CEA. Papyrus uml, 2008. `http://www.eclipse.org/papyrus/`.

[10] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983. ISBN: 0-9613921-0-X.

[11] Laura Perini. Visual representations and confirmation. *Philosophy of Science*, 72(5):913–926, 2005.

[12] Stephen M. Kosslyn. *Image and Mind*. Harvard University Press, 1980.

[13] Jill H. Larkin and Herbert A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1):65 – 100, 1987.

[14] Barbara Tversky. Cognitive principles of graphic displays. In *AAAI 1997 Fall Symposium on Reasoning with Diagrammatic Representations*, 1997.

[15] Daniel Moody and Jos Hillegersberg. Evaluating the visual syntax of uml: An analysis of the cognitive effectiveness of the uml family of diagrams. In Dragan Gasevic, Ralf Lammel, and Eric Wyk, editors, *Software Language Engineering*, volume 5452 of *Lecture Notes in Computer Science*, pages 16–34. Springer Berlin Heidelberg, 2009.

[16] ISO/IEC. Iso/lec standard 24744: Software engineering - metamodel for development methodologies, 2007.

[17] D. Moody. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on*, 35(6):756 –779, nov.-dec. 2009.

[18] T. Harju. Lecture notes on graph theory, 2007. Department of Mathematics. University of Turku. Finland.

[19] David Harel. On visual formalisms. *Commun. ACM*, 31(5):514–530, May 1988.

[20] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, April 1986.

[21] Jacques Bertin. *Semiology of graphics : diagrams, networks, maps*. University of Wisconsin Press, Madison, Wisconsin, 1983.

[22] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):pp. 531–554, 1984.

[23] Gerald Lee Lohse. A cognitive model for understanding graphical perception. *Hum.-Comput. Interact.*, 8(4):353–388, December 1993.

[24] William Winn. Learning from maps and diagrams. *Educational Psychology Review*, 3:211–247, 1991.

[25] P. Bottoni and A. Grau. A suite of metamodels as a basis for a classification of visual languages. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 83 –90, sept. 2004.

[26] M. Auer, T. Tschurtschenthaler, and S. Biffl. A flyweight uml modelling tool for software development in heterogeneous environments. In *Euromicro Conference, 2003. Proceedings. 29th*, pages 267 – 272, sept. 2003.

[27] Mark Minas. Visual specification of visual editors with VisualDiaGen. In John L. Pfaltz, Manfred Nagel, and Boris Böhlen, editors, *Applications of Graph Transformation with Industrial Relevance, Proc. 2nd Intl. Workshop AGTIVE'03, Charlottesville, USA, 2003, Revised and Invited Papers*, volume 3062 of *Lecture Notes in Computer Science*, pages 473–478. Springer-Verlag, 2004.

[28] J. De Lara and Hans Vangheluwe. Using atom3 as a meta-case tool. In *4th International Conference on Enterprise Information Systems (ICEIS), April 2002, Ciudad Real, Spain*, pages 642–649, 2002.

[29] Object Management Group. Diagram definition (dd) 1.0, 2012. `http://www.omg.org/spec/DD/`.

[30] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley Professional, 2003.

[31] F. Fondement. Documentation succincte sur gmf. Technical report, Université de Haute Alsace, 2008. `http://fondement.free.fr/lgl/courses/mde/documentation_succinte_gmf.pdf`.